(12)

Constraining Learning with Search Control

Jihie Kim and Paul S. Rosenbloom
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

June 1993
ISI/RR-93-354

\\P8

93 9 17 047

REPORT DOCUMENTATION PAGE	FORM APPROVED OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1993	3. REPORT TYPE AND DATES COVERED Research Report

4. TITLE AND SUBTITLE Constraining Learning with Search Control	5. FUNDING NUMBERS N00014-92-K-2015
6. AUTHOR(S) Jihie Kim and Paul Rosenbloom	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) USC INFORMATION SCIENCES INSTITUTE 4676 ADMIRALTY WAY MARINA DEL REY, CA 90292-6695	8. PERFORMING ORGANIZATON REPORT NUMBER RR-354
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES) ARPA 3701 Fairfax Drive Arlington, VA 22203	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12A. DISTRIBUTION/AVAILABILITY STATEMENT UNCLASSIFIED/UNLIMITED	12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Many learning systems must confront the problem of run time after learning being greater than run time before learning. This utility problem has been a particular focus of research in explanation-based learning. In past work we have examined an approach to the utility problem has been a particular focus of research in explanation-based learning. In past work we have examined an approach to the utility problem that is based on restricting the expressiveness of the rule language so as to guarantee polynomial bounds on the cost of using learned rules. In this article we propose a new approach that limits the cost of learned rules without guaranteeing an a priori bound on the match process or restricting the expressibility of rule conditions. By making the learning mechanism sensitive to the control knowledge utilized during the problem solving that led to the creation of the new rule-i.e., by incorporating such control knowledge into the explanation--the cost of using the learned rule becomes bounded by the cost of the problem solving from which it was learned.

14. SUBJECT TERMS utility problem, EBL (explanation based learning), search control, Soar and chunking	15. NUMBER OF PAGES 8
	16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: *Prepared in cooperation with...; Trans. of ...;* To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Constraining Learning with Search Control

Jihie Kim and Paul S. Rosenbloom
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, U.S.A.
jihie@isi.edu, rosenbloom@isi.edu

tract

Many learning systems must confront the problem of run time after learning being greater than run time before learning. This utility problem has been a particular focus of research in explanation-based learning. In past work we have examined an approach to the utility problem that is based on restricting the expressiveness of the rule language so as to guarantee polynomial bounds on the cost of using learned rules. In this article we propose a new approach that limits the cost of learned rules without guaranteeing an a priori bound on the match process or restricting the expressibility of rule conditions. By making the learning mechanism sensitive to the control knowledge utilized during the problem solving that led to the creation of the new rule — i.e., by incorporating such control knowledge into the explanation — the cost of using the learned rule becomes bounded by the cost of the problem solving from which it was learned.

1 Introduction

The identification of the utility problem in explanation-based learning (Minton 1988), has prompted considerable research on how to assure — or at least to improve the chances — that learned knowledge which is intended to speed up system performance will in fact do so, rather than slow it down. Our own efforts on the utility problem have focused on two subissues with respect to Soar, an architecture that combines general problem solving abilities with a chunking mechanism that is a variant of explanation-based learning (Rosenbloom, Laird, Newell, and McCarl 1991). The first subissue is the problem of *expensive chunks* (Tambe, Newell, and Rosenbloom, 1990; Tambe and Rosenbloom 1990), in which individual learned rules are so expensive to use that the system suffers a slow down from learning. The second subissue is the *average growth effect* (Doorenbos, Tambe, and Newell 1992; Doorenbos 1993), in which the system learns so many rules — none of which individually need be all that expensive — that a slow down results.

In this article we focus on expensive chunks. The prior work on expensive chunks demonstrated their existence in a number of tasks, identified their origins in the exponential (in the number of rule conditions) upper-bound on the cost of matching individual rules, and investigated a range of possible restrictions on the expressibility of the rules that permit polynomial upper-bounds on match cost. The most successful of these restrictions is *unique-attributes*, in which match cost is bounded by a linear function of the number of conditions. Imposition of the unique-attributes restriction disallows object attributes from having more than one value. Values can be structured objects with many parts, but they cannot be unstructured sets of objects. Figure 1-a shows an unrestricted encoding for part of a state in the blocks world. The attribute *block* of object S1 is not a unique attribute because it has three distinct values. Figures 1-b and 1-c show two different unique-attribute encodings of the same structure.

(S1 ^type state)	(S1 ^type state)	(S1 ^type state)
(S1 ^block B1)	(S1 ^block B1)	(S1 ^focus B2)
(S1 ^block B2)	(B1 ^next B2)	(B1 ^left B1)
(S1 ^block B3)	(B2 ^next B3)	(B1 ^right B3)
(a)	(b)	(c)

Figure 1: Unrestricted (a) and unique-attribute (b-c) encodings in the blocks world.

Although a number of systems have been successfully recoded into unique-attributes, and reaped significant time savings as a result, there are still some outstanding problems with it. In particular, the encoding radically increases the number of rules used in specifying some tasks, and may also require many more rules to be learned to achieve the same level of coverage (that

A-1

is, generality) as was previously attainable by a small number rules.

In this article, we propose an alternative diagnosis for the cause of expensive chunks, along with a new approach for eliminating expensive chunks that is derived from this new diagnosis. The core idea is to focus on the relationship between the problem-space search upon which the learning is based and the search performed, during match, by the rule learned from this problem-space search. In the search of the problem space, some path — that is, some sequence of operators — is followed that eventually leads to a result. The actual path followed usually depends on meta-level control rules that determine which operators are selected for which states. These control rules should affect only the efficiency with which the result is found, and not its correctness. As a result, when a new rule is acquired from a trace of this problem solving, the control rules are not included as part of the explanation of the result. This omission, which turns out to also be the approach taken in PRODIGY (Minton 1993)[1], increases the generality of the learned rules, while it should not affect their correctness.[2]

The problem with this approach, however, is that the learned rules are not now constrained by the path actually taken in the problem space, and thus can perform an exponential amount of search even when the original problem-space search was highly directed (by the control rules). For example, with suitable control knowledge in the Grid Task (Tambe, Newell, and Rosenbloom 1990) it is possible to solve the problem of finding a path between two nodes in time that is linear in the length of the path. However, the rule learned from this search may be so general that, when it matches, it searches over all paths of that length. This rule is quite general, as it can solve any problem that has a solution of that length; however, this generality is only obtained at an enormous cost (i.e., the cost is exponential in the length of the path).

The solution suggested by this diagnosis is to incorporate traces of the control rules utilized in the problem-space search into the explanation of the result. This should enable the match process for learned rules to focus on just the precursors for the path that was actually followed, and thus ensure that the match process for a learned rule is bounded in complexity by the problem-space search from which it was learned. Because the match process runs at a faster rate than the problem solving process, this should solve the expensive chunks problem by ensuring that using the learned rule takes no more time than was taken by the original search.

This approach is closest in spirit to that taken in (Shell and Carbonell, 1991). In that work, iterative paths found during problem-space search resulted in the addition of iterative constructs to the macro-operators acquired from the search. These iterative macro-operators are then used in a way that guarantees that they take the same path followed in the problem space. Shell and Carbonell claim that their approach solves the expensive chunks problem. However, it doesn't completely because not all expensive chunks arise from iteration. Our approach captures the same basic intuition, but in a manner that it is both more general and simpler. It is more general because it captures the factors that determined the entire path, rather than just the iterative portions, and thus handles all of the causes of expensive chunks. It is simpler because it does not require an enhanced macro-language or special purpose mechanisms for detecting iteration. Instead, it simply expands by a small amount the content of the explanation used during learning.

In contrast to our earlier approaches to expensive chunks, this new approach ***imposes no expressibility limitations on the encoding of tasks***. On the positive side, this means that the problem of expensive chunks can be solved without increasing the difficulty of task encoding. On the negative side, this means that no sub-exponential bound is being imposed on the match process — if the original rules encoded into the system require exponential matches, then so may the learned rules. We have thus effectively split off the goal of removing expensive chunks from the related goal of guaranteeing bounds on the match, and in the process found a weaker approach that solves the former but not the latter, but with no limit on task expressibility.

Despite this result, this new approach is not free of problems. One significant problem is that it doesn't specify what to do when decisions in a search are based on *lack of knowledge*. In such circumstances, the learning process has no explanation for why a choice was made, and therefore can acquire rules that are just as expensive as those learned by the unaltered learning mechanism. The other significant problem is that, as with unique-attributes, this approach can lead to learned rules that are less general than would be acquired by the unaltered learning mechanism. This comes about here, not because of limitations on the representation, but because additional conditions are incorporated into learned rules based on control rules that are now part of the explanation. These conditions provide efficiency, but at the cost of eliminating search that otherwise would allow the rules to apply in more circumstances.

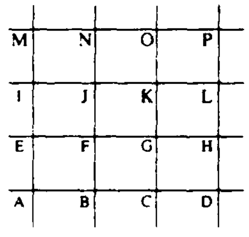
[1]In Prodigy, selection and rejection rules are included in the explanation, but preference rules are not. Likewise, Soar currently also includes require and prohibit preferences, but not desirability preferences.

[2]In Soar, this actually can at times affect correctness, but the discussion of this will be postponed to the final section.

In the next section we look at this first, lack-of-knowledge problem in more detail, and identify two possible solutions. In the subsequent section we present experimental results from using the new approach to expensive chunks in combination with one of the proposed solutions to the lack-of-knowledge problem, in particular, a solution that depends on a novel restriction on the expressiveness of the resulting system. In the process we will discuss the impact of the second, specialization-of-learned rules problem. The final section summarizes and discusses issues for future work.

2 Decisions Based on Lack of Knowledge

In Soar, a real lack of knowledge — as reflected in an insufficient set of *preferences* about a decision — leads to an impasse rather than to a decision. Thus it might seem that Soar wouldn't suffer from this problem. However, it does have a construct — an *indifferent* preference — that allows the explicit statement of indifference among a set of choices. The decision procedure is then free to select randomly among the indifferent choices. The resulting choice is thus made in such a way that no explanation of the selection among the indifferent alternatives is possible based just on the initial situation.



```
(F ^connected B)
(F ^connected E)
(F ^connected J)
(F ^connected G)
(G ^connected F)
```

```
(sp operator-goto
    (goal <g> ^problem-space <p>
              ^state <s>)
    (<p> ^name grid-task)
    (<s> ^at <loc1>)
    (<loc1> ^connected <loc2>)
    -->
    (<o> ^name goto-loc
         ^at <loc1> ^to <loc2>)
    (<g> ^operator <o>))
```

(a) (b)

Figure 2: The Grid Task.

Consider an example from the Grid Task — a problem known to lead to expensive chunks (Tambe, Newell, and Rosenbloom 1990) — shown in Figure 2-a. The problem is to go from point F to point P, a path of length four. Because point F is connected to four adjacent points, four operators are suggested by rule *operator-goto* (Figure 2-b)[3]. Since the knowledge required to choose among them is not directly available in productions, an impasse occurs on operator selection. In the subgoal created for this impasse, Soar normally employs the *selection* problem space, which contains *evaluate* operators that can be applied to the competing task operators. Once generated, these evaluations will be turned into preferences that allow one of the task operators to be selected. However, the system has no direct knowledge about which of the four operators it ought to evaluate first, so without further assistance it would impasse again, and possibly continue this recursive subgoaling indefinitely. To avoid this, one of Soar's general background rules generates indifferent preferences for the set of evaluate operators. This lets it pick one at random, and begin to make progress.

[3]Symbols enclosed in angle brackets are variables.

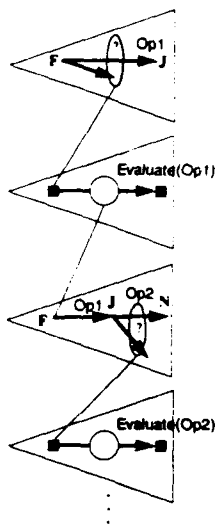


Figure 3: Problem solving in the Grid Task.

If, as is often the case, the information about how to evaluate an operator is not directly available, an evaluation subgoal (to implement the evaluate operator) is created. The task in this third-level subgoal is to determine the utility of the operator. To do this, it performs a bit of lookahead search, trying out the task operator (possibly in simulation) on the original task state. If the resulting state can be evaluated, then the subgoal terminates, otherwise the process continues, recurring on the question of what task operator to apply to this new state. Figure 3 shows this search process in the Grid Task which continues until the point P is reached.

In this overall lookahead search, indifferent preferences indirectly determine which path the system moves down, by directly determining which of the operators are evaluated at each point. However, the rules learned from this search can gather no explanation from the indifferent preferences as to why one path was taken rather than another. Figure 4 shows such a learned rule. This rule says that if you are at location <l1>

```
(sp chunk-example
   :chunk
   (goal <g> ^problem-space <p> ^state <s>
             ^operator <o> + ^desired <d>)
   (<o> ^name goto-loc ^at <l1> ^to <l2>)
   (<p> ^name grid-path)
   (<s> ^at <l1>) (<d> ^at <l5>)
   (<l2> ^connected <l3>) (<l3> ^connected <l4>)
                          (<l4> ^connected <l5>)
   -->
   (<g> ^operator <o> >))
```

Figure 4: An expensive chunk learned from indifferent choices.

and want to get to location <l5>, and there is an operator that takes you from <l1> to <l2>, and there is a connected path from <l2> to <l5> (via two intermediate points, <l3> and <l4>), then the operator is the best choice. This rule is expensive because it may need to search an exponential number of paths of length four to find one that has this property. Even if the original problem-space search happened to locate the correct path by accident on its first try, or if outside guidance was provided to lead it down the correct path, the resulting rule would still incorporate this exponential search.

There are (at least) two possible ways of solving this problem. The first is to alter the learning and match processes so that they more appropriately reflect the semantics of indifferent preferences. Use of an indifferent preference means that a random selection of a single path should be made. However, the match algorithm always follows all paths. So, reflecting the semantics of indifference should involve altering the learning and match processes so that use of indifferent preferences during problem-space search yields the random choice of a single alternative during the corresponding part of the match of the learned rule. If, in fact, the indifferent preference meant that the system really didn't care which of the paths was taken, then any random selection made by the matcher should be as good as any other. If, however, the indifferent preference actually signified lack of knowledge about the correct path, and not all paths actually do lead to success, then the match will follow one path randomly, and thus will succeed only stochastically.

This first direction looks pretty interesting. It solves the problem without introducing an expressibility limitation, while at the same time introducing a stochasticity into the use of learned rules, and a resulting gradualness in performance improvement that may be quite useful in modeling human cognition. However, it requires a significant enough alteration in the basic architecture of Soar, that we have decided to first investigate a simpler alternative, and leave this one for future work.

The second way of solving the problem, and the one underlying the results reported here, is to disallow the use of indifferent preferences. Their ability to select randomly among alternatives is then replaced by explicit default orderings on the alternatives. If there are any substantive reasons why one alternative should be selected ahead of another, they can be incorporated into this ordering. To the extent that there are no substantive reasons, an arbitrary ordering can be imposed. The key, though, is that these orderings are generated explicitly by rules that distinguish among the alternatives, and therefore leave behind a trace that can be used in explaining why one alternative is picked over the others. This may not provide a "good" explanation, in the sense of capturing a suitable level of generality to support transfer to related situations; however, it will at least be sufficient to distinguish the one selected alternative from the others during the match, and thus to make the resulting learned rules cheap.

For the Grid Task, an arbitrary ordering of the operators can be assigned according to the direction of movement. For example, first *up*, then *down*, then *left*, and finally *right*. It is important to note that this ordering is just used in place of the indifferent preferences on the evaluate operators in the selection space. Thus it determines the order in which the operators are evaluated, but does not dictate an ordering on the task operators. This latter ordering is still to be learned, as a new set of control rules, from the lookahead search.

The elimination of indifferent preferences amounts to a limitation on the system's expressibility, though of a form quite different from those previously investigated. It also clearly may impact the generality of the resulting rules, at least to the extent that arbitrary orderings are imposed. As such, it needs to be evaluated, just as was the unique-attributes restriction, in terms of the trade-offs it provides among expressibility, speed, and generality.

3 Experimental Results

In this section we look at how well the incorporation of search control into learned rules, in combination with the elimination of indifferent preferences, compares with both an unaltered version of Soar and a unique-attributes version. The results are all from Soar6 (version 6.0.3), the latest C-based release of Soar (Doorenbos 1992), which is approximately 10-40 times faster than Soar5 (the previous Lisp-based release). The experimental version is just like the standard system, except that the explanations upon which new rules are based incorporate traces of the control rules that determined the choices made in problem solving. In particular, the system computes the minimum set of preferences sufficient to determine each choice that was made, so that if the set of preferences overdetermines the choices, the redundant preferences (and their rule traces) are pruned from the explanation to make the created rule as general as possible.

Grid Task	Average CPU Time (sec)	
	Before Learning	After Learning
Original	5 38	24 79
Search control	6 83	1 18
Unique-attribute	6 82	0 95

Table 1: Average CPU time in the Grid Task.

```
(sp chunk-search control
    chunk
    (goal <g> ^operator <r> + ^operator <u> +
       ^operator <d> + ^desired <d1> ^state <s>
    (<r> ^priority 4 ^at <l1> ^to <l2>)
    (<u> ^priority 3) (<d> ^priority 1)
    (<s> ^at <l1>) (<d1> ^at <l8>)
    (<l2> ^right <l3> ^connected <l3>
          ^down <l4> ^connected <l4>
          ^up <l5> ^connected <l5>)
   (<l3> ^up <l6> ^connected <l6>
          ^down <l7> ^connected <l7>)
   (<l6> ^connected <l8> ^up <l8>)
   -->
   (<g> ^operator <r> >))
```

(a)

```
(sp chunk-unique-attribute
    chunk
    (goal <g> ^operator-right <r> +
       ^desired <d> ^state <s>
    (<r> ^at <l1> ^to <l2>)
    (<s> ^at <l1>) (<d> ^at <l5>)
    (<l2> ^right <l3>)
    (<l3> ^up <l4>)
    (<l4> ^up <l5>)
    -->
    (<g> ^operator <r> >))
```

(b)

Figure 5: Chunks from search-control and unique-attribute versions.

Table 1 shows the average CPU time per problem (in seconds) for the three versions, across seven different problems in the Grid Task, both before and after learning. All of the grid problems used here are searches for paths of length six; for example, in Figure 2-a, a problem to go from point A to point P is a length-six problem. For experimental efficiency, the results shown here also assume a 10x10 bounded grid instead of the unbounded grid in Figure 2-a. The first row in Table 1 shows the times before and after learning for the unaltered version of Soar. Without including search control in chunking, or restricting the task representation, the time after learning is greater than the time before learning for all these problems (by an average factor of 4.61), and for one of the problems it is more than a factor of nine greater. This is true even though the number of problem solving steps (i.e., decisions) is decreased via learning from 133 to 8. This extra cost is directly attributable to the large amount of time spent matching expensive chunks.

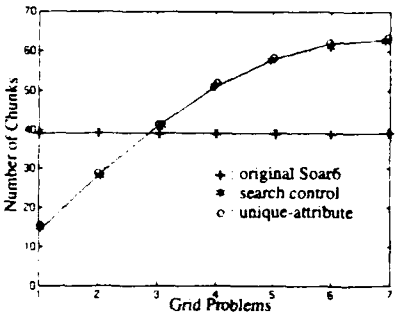


Figure 6: Number of accumulated chunks in the Grid Task.

The second and third rows in Table 1 show the corresponding CPU times for the search-control and unique-attributes versions of the Grid Task.[4] Both show more than a factor of five reduction in execution time after learning. In each problem, they show essentially the same pattern: the time after learning is a small constant value that is uniformly less than the time before learning. This implies that both have solved the expensive chunks problem for this task.

The extra time before learning in the search-control and unique-attribute versions stems from the increase in tokens brought about the additional rule conditions that discriminate among moving directions, as shown in the conditions of the chunks in Figure 5. These two chunks correspond to the expensive chunk in Figure 4. The difference in run times after learning between the search-control version and the unique-attribute version in Table 1 is also due to the extra conditions in the search-control-version chunks. However, this yields only a minor effect, as analyzed in (Tambe 1991).

Figure 6 shows the cumulative number of chunks acquired while solving the eight Grid-Task problems. The unmodified version of Soar learned general enough chunks from the first problem to cover all of the other length-six problems. The other two approaches needed to learn additional chunks for each new problem. In these problems, both learned the same number of rules with the same generality. Although there are additional contraints induced by the extra conditions in Figure 5-(a), both chunks in Figure 5 have the same generality in that they describe the same grid path followed by the lookahead search to reach the desired point, and nothing more than that.

[4]The unique-attribute representation replaces the multi-attribute *^connected* with four distinct attributes *^up*, *^down*, *^left* and *^right*.

Eight-puzzle Task	Average CPU Time (sec)	
	Before Learning	After Learning
Original	5.31	9.75
Search control	5.30	1.55
Unique-attribute	5.72	1.21

Table 2: Average CPU time in the Eight-puzzle Task.

Table 2 compares these three methods on the Eight-puzzle Task — another task known to produce expensive chunks (Tambe, Newell and Rosenbloom 1990). In the multi-attribute representation, a state points to nine bindings (using attribute ^binding), each of which connects a cell from the static 3x3 structure of the board to a tile. For example, in (B1 ^cell C1) (B1 ^tile T1), binding B1 connects cell C1 to tile T1. A cell points to all of its neighboring cells. For example, (C1 ^next C2),(C1 ^next C3), and so on.

The search-control version used here distinguishes operators by the direction that they move the blank cell: down, up, left and right. The unique-attribute representation removes the multi-attribute ^*binding* by numbering it, ^*binding1*, ^*binding2*,. . . ., ^*binding9*, and replaces ^*next* with 4 attributes ^*down*, ^*up*, ^*left* and ^*right*. As shown in Table 2, the time after learning is less than the time before learning in the search-control and unique-attribute versions, while the original Soar requires more CPU time after learning. As in the Grid Task, both the search-control and unique-attribute versions have eliminated the expensive chunks that occur in the unmodified version. The difference in run times between the search-control version and the unique-attribute version stems from the same reason as in the Grid Task.

The number of rules used to encode the Eight Puzzle in the three versions tells an interesting story. The original version of Soar uses 13 rules, the search-control version uses 16 rules, and the unique-attributes version uses 93 rules. The small growth in going from Soar to the search-control version stems from the need to differentiate and provide a default order on the domain operators. If there are n possible operators, n-1 additional rules are required. The large growth in the unique-attributes version stems from the need to create rules for each specialization of a generic attribute into a more specialized unique-attribute. In general, if there are n tests of multi-attributes in a rule, each with m possible specializations, then the unique-attributes version will need to substitute m^n specialized rules.[5]

Figure 7 shows the cumulative number of chunks learned while solving the nine Eight-puzzle problems.

[5]There are ways to reduce this number by splitting these tests across a sequence of rules, but that approach also has its own problems.

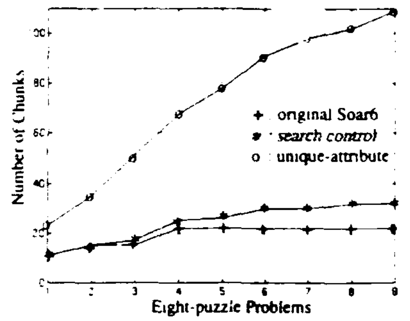


Figure 7: Number of accumulated chunks in the Eight-puzzle Task.

The search-control version required 32 new rules, as compared to 109 new rules for the unique-attribute version, and 22 new rules for the original version. Thus, for this task, the search-control version is quite close to the original Soar version, and both show a distinct advantage over unique-attributes.

Unique-attributes' need here for many new rules stems from the same source as its large number of initial rules, plus the following fact. The search-control version need not distinguish between values of a multi-attribute as long as it doesn't affect the decision that is based on lack of knowldge, while the unique-attribute version must replace the muti-attribute anyhow. Attribute ^*binding* in the above Eight-puzzle task is an example of a multi-attribute that doesn't affect the decision based on lack of knowledge.

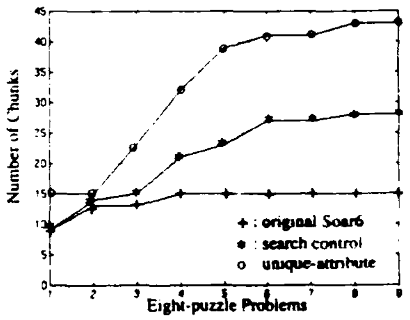


Figure 8: Number of chunks in different Eight-puzzle representation.

The Eight Puzzle can also be expressed via a different set of rules, without the multi-attribute ^*binding*. Although there is considerable reduction in the number of rules, Figure 8 shows that the unique-attribute version still needs more rules because of the former effect.

4 Summary and Discussion

Unique-attributes solve the expensive chunks problem by restricting the expressiveness of rules down to where the match can be guaranteed to run in polynomial (in particular, linear) time. This provides strong assurances about system performance, but also negatively impacts task creation and learned-rule generality. Here we have proposed and investigated a new approach — based on including search-control in the explanations upon which new rules are based — that solves the expensive chunks problem, but not by enforcing a fixed computational bound on the match process. Instead, the complexity of the match of a learned rule is bounded by the complexity of the search from which it was learned. This gives up an overall guarantee on system performance, but given an initially encoded system, learning will not make it worse. In exchange for this weakening of the guarantee, this new approach shows potential for ameliorating both of the negative side effects introduced by unique-attributes.

One additional positive side-effect of the search-control approach is that it removes one possible source of overgeneralization in Soar (Laird, Rosenbloom, and Newell 1986). Though search control is not supposed to affect the correctness of results generated in problem spaces, it sometimes unavoidably does. In situations in which results are returned from a problem space before the goal test succeeds, or where the goal test is itself overgeneral, search control may play an influential role in determining the correctness of the result. Under such circumstances, the current approach — not including this search control in the explanation process — can yield overgeneral learned rules. However, by including this search control into the explanation of the result, the proposed approach removes this potential source of overgenerality.

A possible negative side-effect of the search-control approach is that it increases the difficulty of directing the reconstruction process that underlies Soar's approach to knowledge-level learning (Rosenbloom, Laird, and Newell 1987; Rosenbloom and Aasman 1990). There we took advantage of search control's absence from explanations in learning a rule whose actions mirrored some perceived object structure, but whose conditions did not test the perceived object. With this option no longer available, a new approach must be employed. One possibility that was actually already under investigation independently of this work, is a form of *situated reconstruction*, in which reconstruction is guided by features of the immediate situation other than those to be reconstructed (Vera, Lewis and Lerch 1993).

In addition to investigating options for knowledge-level learning, several other issues need near-term attention. At the top of the list is extending the experimental results to a wider range of tasks — both those that traditionally yield expensive chunks and those that don't — and to quantitative analyses of speed ups and (losses of) generality. Also useful would be a theoretical analysis of the method, and of its potential to avoid (or lead to) slow downs with learning. There is also a subtle issue that needs to be addressed that only occurs when there are more options available at performance than at learning time; in particular, if the conditions learned to discriminate among the options available at learning time are not sufficient to discriminate among these new options, additional match search may be introduced. Finally, altering the architecture so as to permit the appropriate use of indifferent preferences would enable the removal of the one expressibility limitation that it was found necessary to impose.

Acknowledgments

This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan as part of contract N00014-92-K-2015 from the Defense Advanced Research Projects Agency (DARPA) and the Naval Research Laboratory (NRL).

Thanks to Milind Tambe and Soar group in USC-ISI for valuable comments on earlier draft of this paper, and to Bob Doorenbos for useful hints on Soar6 source code.

References

Doorenbos, B., Tambe, M., & Newell, A. (1992). Learning 10,000 chunks: What's it like out there? *Proceedings of the Tenth National Conference on Artificial Intelligence*. (pp. 830-836).

Doorenbos, B. (1992). Soar6 release notes.

Doorenbos, B. (1993). Matching 100,000 learned rules. *Proceedings of the Eleventh National Conference on Artificial Intelligence*. (to appear).

Laird, J., Rosenbloom, P., & Newell, A. (1986). Overgeneralization during knowledge compilation in Soar. *Proceedings of the Workshop on Knowledge Compilation*. (pp. 46-57).

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. *Proceedings of the Seventh National Conference on Artificial Intelligence*. (pp. 564-569).

Minton, S. (1993). Personal Communication.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1987). Knowlege level learning in Soar. *Proceedings of the Sixth National Conference on Artificial Intelligence*. (pp. 499-504).

Rosenbloom, P. S. & Aasman, J. (1990). Knowledge level and inductive uses of chunking (EBL). *Proceedings of the Eighth National Conference on Artificial Intelligence*. (pp. 821-827).

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence 47* (pp. 289-325).

Shell, P. & Carbonell, J. (1991). Empirical and analytic performance of iterative operators. *The 13th Annual Conference of The Cognitive Science Society.* (pp. 898-902).

Tambe, M. (1991). Eliminating combinatorics from production match. PhD thesis, Computer Science Department, Carnegie Mellon University.

Tambe, M., Newell, A., & Rosenbloom, P. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, Vol. *5*, (pp. 299-348).

Tambe, M., & Rosenbloom, P. (1990). A framework for investigating production system formulations with polynomially bounded match. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 693-700).

Vera, A., Lewis R. L., & Lerch F. J. (1993). Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.* (to appear).